

# ReactJS PoC for Product Engineering role

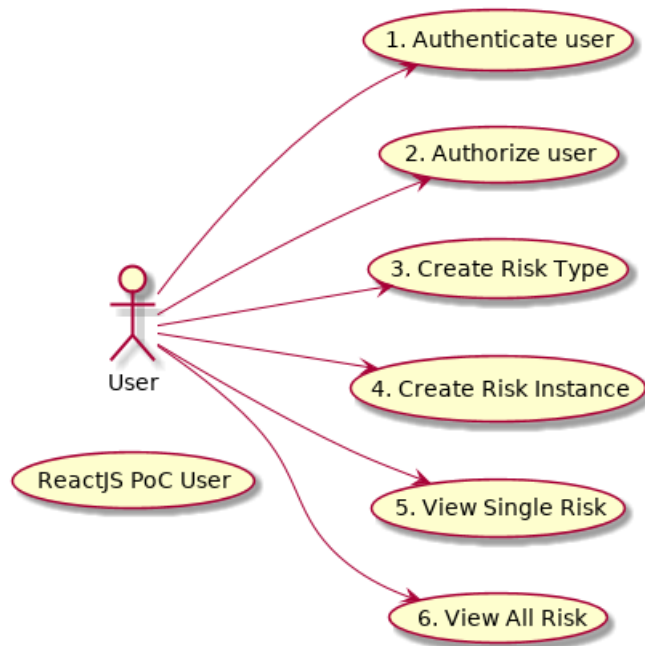
---

## Table of Contents

Use case Diagram for ReactJS PoC.....	2
Authenticating user in ReactJS PoC .....	3
Use case - Authenticating user. ....	3
Sequence diagram for authentication user in ReactJS PoC .....	4
Description for authenticating user in ReactJS PoC.....	5
View Single Risk screen in ReactJS PoC.....	6
Use case – View Single Risk.....	7
Class diagram View Single Risk. ....	8
Sequence diagram View Single Risk.....	10
Description for fetching data for View Single Risk screen in ReactJS PoC .....	11
Message flow within various classes in typical React & Redux based application.....	12
Message flow Diagram.....	12
References .....	13
ReactJS .....	13
Redux .....	13
Element React.....	13
Create React App .....	13
CoreUI for React.....	13
Enzyme.....	13
ReactJS and redux integration .....	13
ReactJS children composition patterns.....	13
ReactJS Redux testing using Jest and Enzyme .....	13

## Use case Diagram for ReactJS PoC

This UI Web application fulfils following use cases.



To keep this documentation short I will elaborate on Authentication and View Single Risk use cases with Class diagrams and Sequence diagrams to show which classes are involved and how these classes communicate with each other to realize the respective use case.

Similar set of classes are used to realize View All Risks as well as Create Risk Type and Create Single Risk Instance.

# Authenticating user in ReactJS PoC

Above use case is fulfilled by Login Page in ReactJS PoC

## Use case - Authenticating user.

**DESCRIPTION:** When user access ReactJS PoC system will ask user to enter valid credentials

**ACTORS:** ReactJS PoC User

**PRECONDITIONS:** Latest Chrome or Firefox browser, Internet connection. App does work on Mobile devices. Though extensive testing has been done only for Personal Computer.

**POSTCONDITIONS :** User is presented with ReactJS PoC Dashboard. Dashboard briefly explains the functionality of each screen in the application. i.e. Create Risk Type, Create Risk Instance, View Single Risk and View All Risk. Depending on if user is Admin user or not User icon on Top right icon reflect what kind of Role user has with this system. It also mentions role he/she has in text form (Roles: Admin / Editor)

**BASIC FLOW OF EVENTS :** User enter user name and password in login screen. Some basic form validation has been done in this screen which prevents any password less than 5 characters. User can show hide password he / she is inputting to make sure correct password is submitted for validation. Upon successful login he / she is directed to Dashboard screen.

**EXCEPTIONAL FLOW :** In case there is no internet connectivity or if DRF API does not respond in certain timeframe. Login screen will flash error message indicating Login attempt failed. This UI Web application and DRF API has been deployed using Free Heroku account. It is quite possible that due to inactivity Heroku dyno may be down at that moment and application failed to respond in stipulated time. When Login attempt fails, user should try to Login again after 2/3 minutes

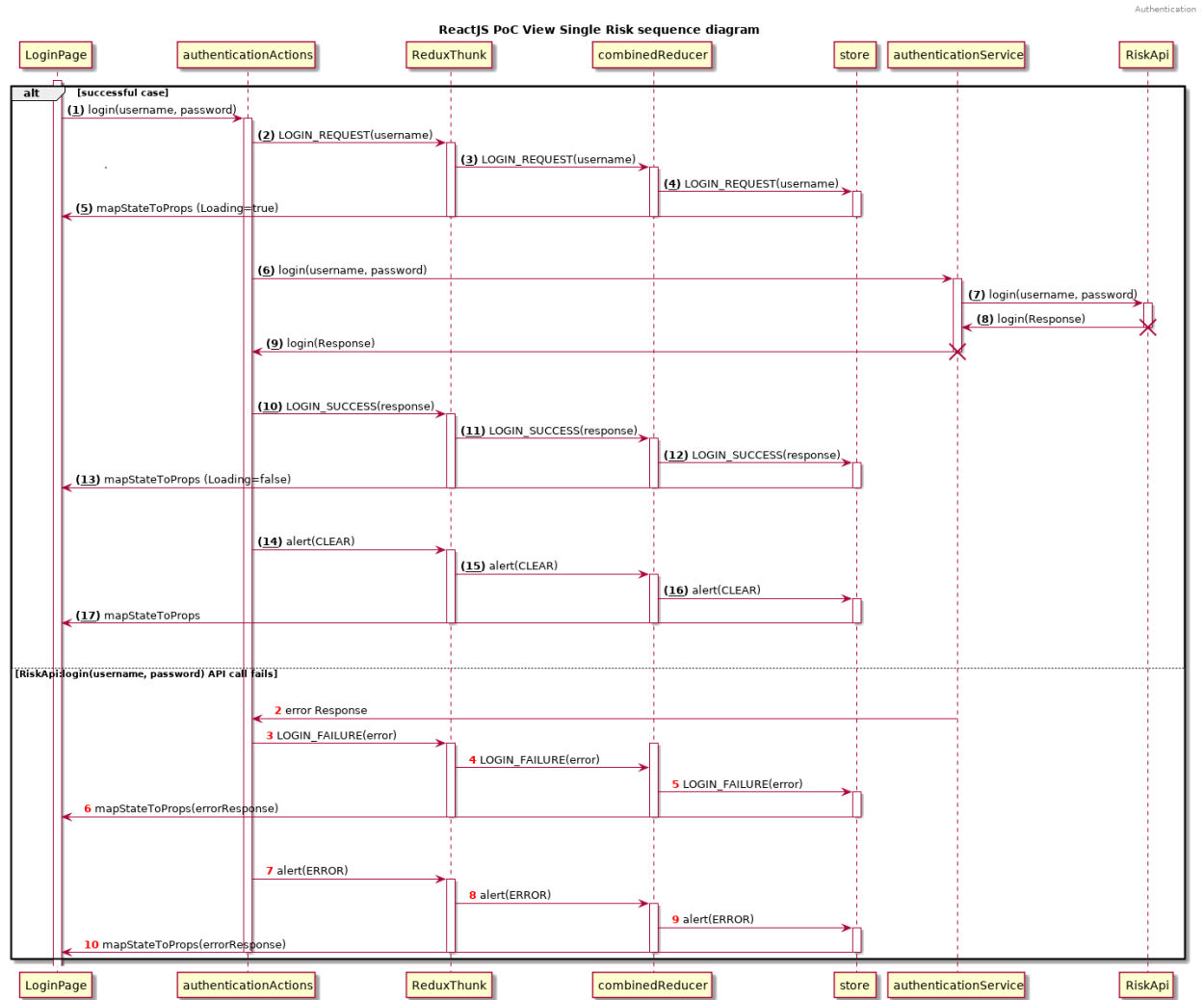
### KEY SCENARIOS :

1. Login attempt successful and user redirected to DashBoard screen.
2. Login attempt failed because credential provided were wrong. Screen shows alert message at the bottom of screen.
3. Login attempt failed because Heroku dyno was not up. Screen shows alert message at the bottom of screen.

**SPECIAL REQUIREMENTS:** ReactJS PoC User input login name and password. UI Web application works in conjunction with Django Rest based Framework henceforth called DRF API will validate the credential provided. Both these systems should be deployed and should be up and running.

## Sequence diagram for authentication user in ReactJS PoC

Please also see message flow within various classes in Redux based application at the end of this document.



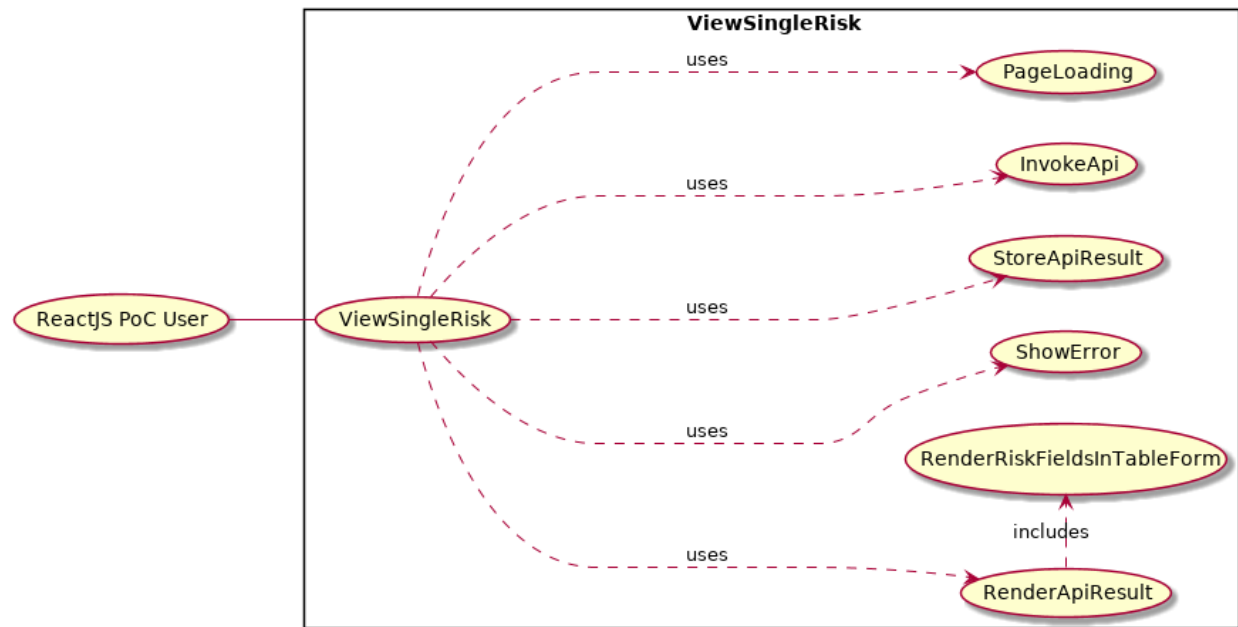
## Description for authenticating user in ReactJS PoC

Here are the following participants in Authentication use case.

LoginPage, authenticationActions, ReduxThunk, combinedReducer, Redux store, authenticationService, RiskApi

1. **Login page** invokes bound method of LoginPage to dispatch to Login method to **authenticationActions**.
2. **authenticationActions:Login** method dispatch LOGIN\_REQUEST to ReduxThunk.
3. **ReduxThunk** in turn send LOGIN\_REQUEST action to combinedReducer.
4. **combinedReducer** dispatch LOGIN\_REQUEST to store which invokes matching action implemented by authentication reducer it updates store and set *Loading* flag to true which is used by UI to show page loading image.
5. **authenticationActions** further dispatches invokes Login method of **authenticationService** which returns promise. Upon successfully resolving this promise authenticationActions:Login method further dispatch **LOGIN\_REQUEST\_SUCCESS**, #2 to #4 repeats for **LOGIN\_REQUEST\_SUCCESS** which is used to set LoggedIn flag to true in ReduxStore.
6. **authenticationActions:Login** method further dispatch **alert(CLEAR)** action , #2 to #4 repeats for **alert(CLEAR)** which is used to clear error flag in ReduxStore. Error flag is used by **ToggleContainer** reusable component to clear out any **Alert** message shown to user in case any **RiskApi** call fails.
7. If **RiskApi** call fails due to invalid credentials or network error message flow take place as per error scenario mentioned in Sequence diagram.
8. If promise errors out **authenticationActions:Login** method further dispatch **LOGIN\_REQUEST\_FAILURE**, #2 to #4 repeats for **LOGIN\_REQUEST\_FAILURE** which is used to set Error flag and message in ReduxStore. **ToggleContainer** uses error flag and error message to show Alert message to indicate Api method failure.

## View Single Risk screen in ReactJS PoC



## Use case – View Single Risk.

**DESCRIPTION:** User can select Risk Name from dropdown box and system will show Risk Name and associated Risk Fields on dynamically populated form

**ACTORS:** ReactJS PoC User

**PRECONDITIONS:** Latest Chrome or Firefox browser, Internet connection. App does work on Mobile devices. Though extensive testing has been done only for Personal Computer. User has successfully logged into the ReactJS PoC. User is able to Navigate to View Single Risk form from Sidebar. User can select Risk Name from Select Risk dropdown box displayed on form.

**POSTCONDITIONS :** User is presented with ReactJS PoC View Single Risk form. View Single Risk form is displayed which shows Risk Name and description plus dynamically populated controls corresponding to Risk Fields associated with Single Risk. Thus page will render either text input or date picker or integer input or currency input control or float control in read-only mode to show field value per Risk Field

**BASIC FLOW OF EVENTS:** User navigates to View Single Risk page. User selects Risk Name from Select Risk dropdown box displayed on form. View Single Risk form is displayed which shows Risk Name and description plus dynamically populated controls corresponding to Risk Fields associated with Single Risk. Verify that while fetching data from Backend user is presented with Page Loading icon. Since data is being fetched asynchronously. Once data is available Page loading icon disappears and User is able to view various controls corresponding to Risk Fields of that Risk.

**EXCEPTIONAL FLOW :** In case there is no internet connectivity or if DRF API does not respond in certain timeframe. View Single Risk screen will flash error message indicating fetch attempt failed. This UI Web application and DRF API has been deployed using Free Heroku account. It is quite possible that due to inactivity Heroku dyno may be down at that moment and application failed to respond in stipulated time. Fetching Single Risk data attempt fails, user should try to fetch again after 2/3 minutes

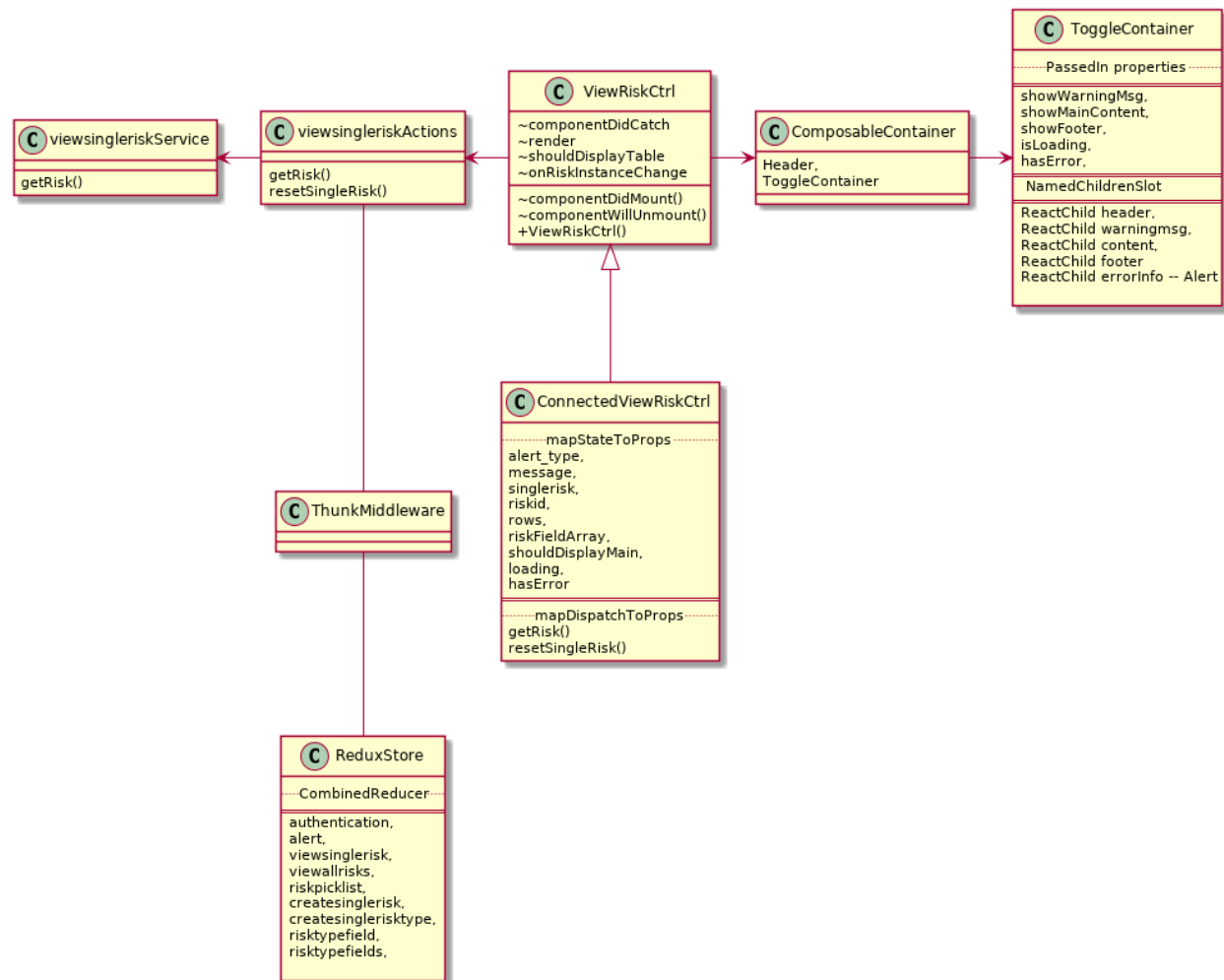
### KEY SCENARIOS:

1. User is able to navigate to View Single Risk screen.
2. User selects Risk name from Drop down box populated with Risk Names entered in System in past.
3. For very brief period of time user is shown page loading icon on screen. Also disable Select Risk Name drop down box.
4. Once data is available screen will hide page loading icon and is able to see all Risk Fields along with Risk Name and description. Enable Select Risk Name drop down box. This will prevent race conditions in fetching and populating dynamic controls.

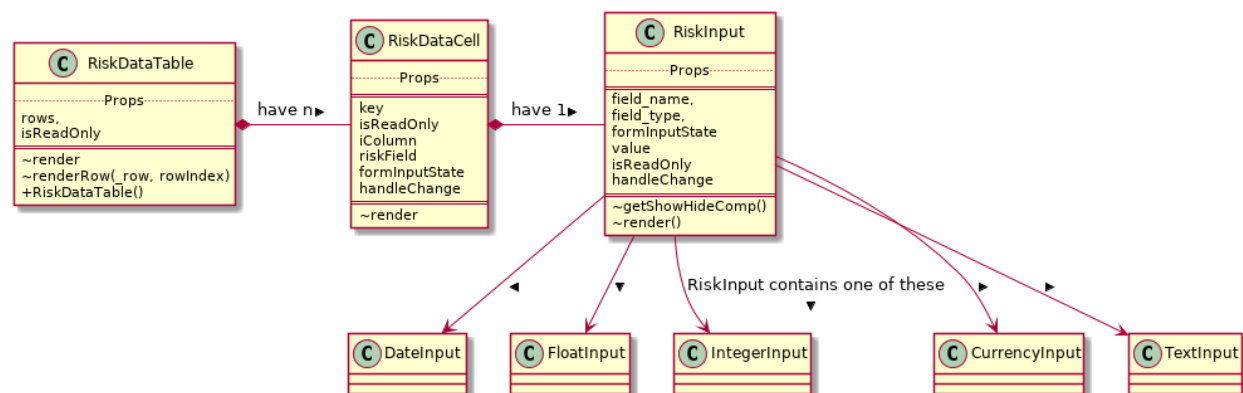
**SPECIAL REQUIREMENTS:** ReactJS PoC User input login name and password. UI Web application works in conjunction with Django Rest based Framework henceforth called DRF API will validate the credential provided. Both these systems should be deployed and should be up and running.

## Class diagram View Single Risk.

Here are the main classes which are used to fetch Single Risk data and populate redux store



Classes related to rendering dynamic controls on form corresponding to Risk Fields in Single Risk



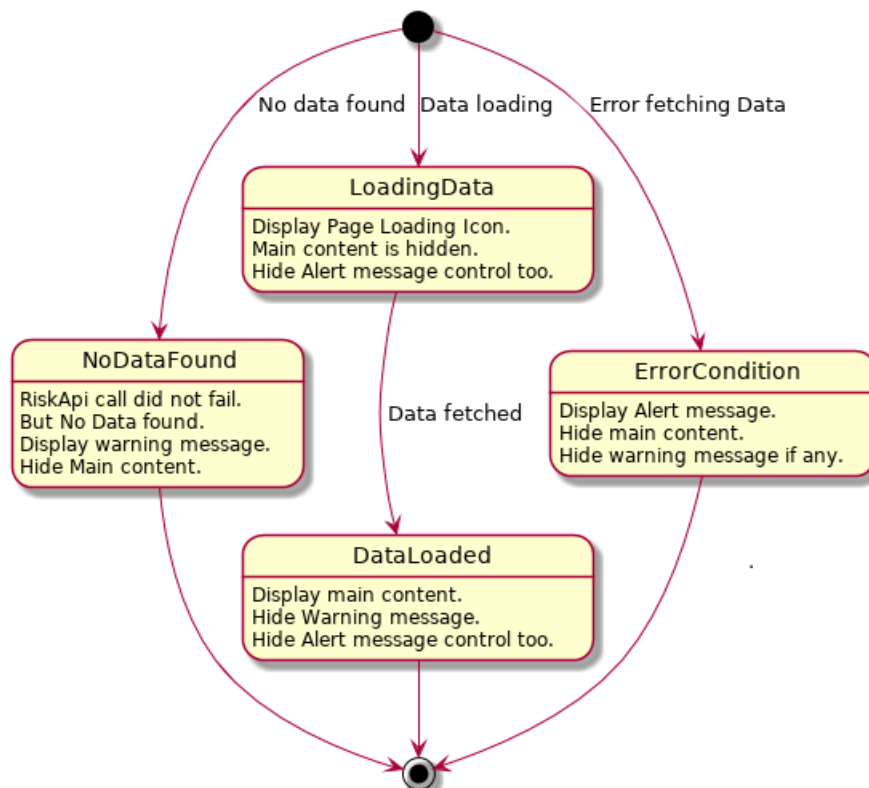


Above Class diagram certainly needs some explanation. All four pages in the ReactJS PoC have some common use cases like once data is fetched and shaped by redux reducer. There can be cases where data is not available example no risks found for given Risk Type. Show page loading icon on screen while data is fetched. Show main content of the form once data is loaded and shaped. In case of any errors while fetching data network error or API level error has to be displayed to user. These generic UI requirements are repetitive. To facilitate some sort generic functionality was needed. Since ReactJS as such does not support generics or some sort of templates like STL in C++. Something close to it could be achieved using Typescript and typescript support by ReactJS, especially `ReactChild` and `NamedChildrenSlots`. `ToggleContainer` as name suggests it toggles its content based on properties passed to it by container component which could be `ViewSingleRiskCtrl` / `ViewAllRiskCtrl` / `CreateSingleRisk` etc. These components can easily derive property value needed by `ToggleContainer` within its `mapStateToProps` function.

Thus various states that `ToggleContainer` support are as follows.

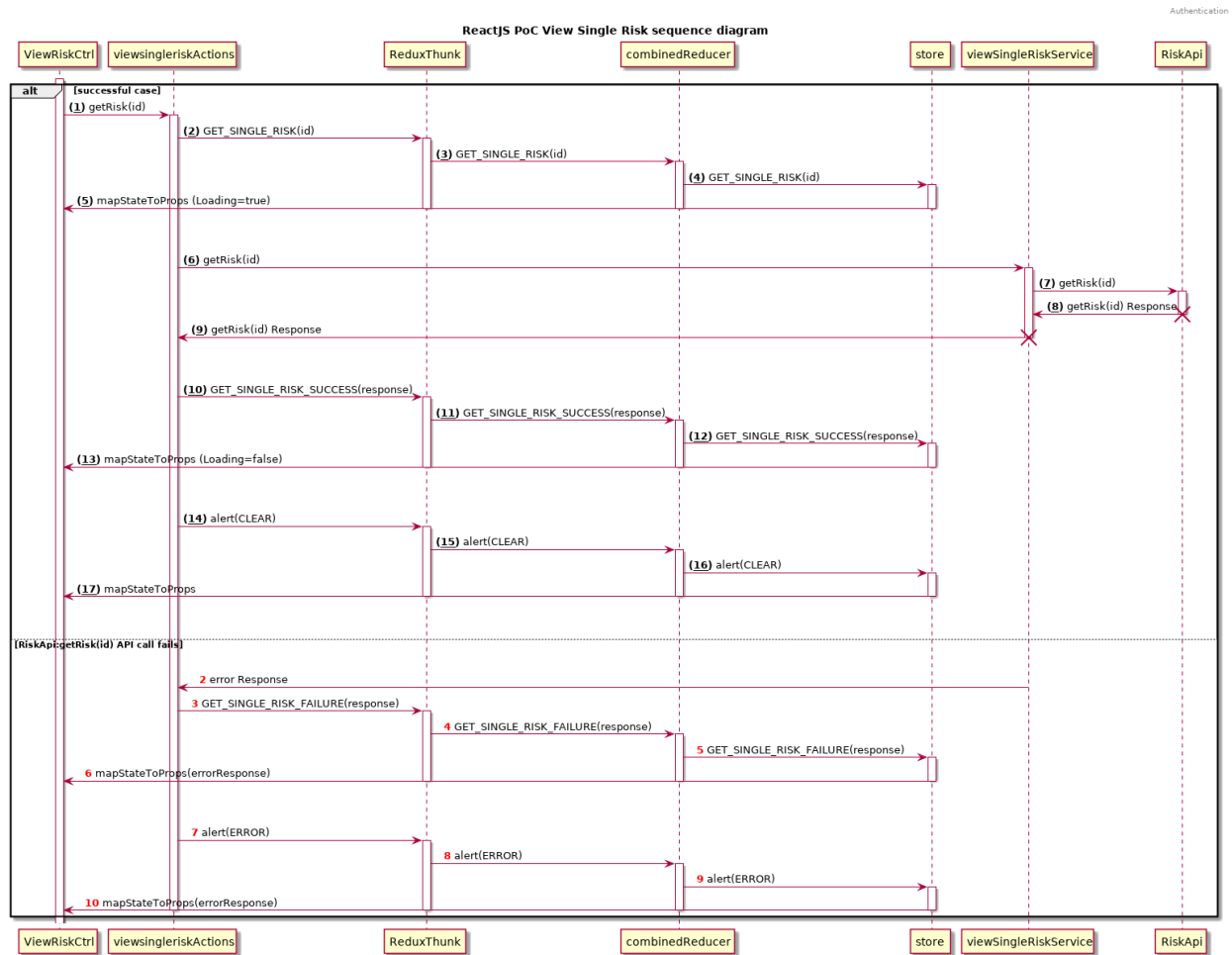
1. Show Warning message when there are no data fetch errors but no data found for given API request.
2. Show page loading icon when data is being fetched
3. Show page content when data is available and no error occurred during API call.
4. Show error message if API call fail for any reason.

Here are various states `Toggle Container` takes.



To complete above explanation ViewRiskCtrl pass Form control containing Risk Name & description along with RiskDataTable for Risk Fields as its main content to ToggleContainer and ToggleContainer takes care of above state change thus facilitating code reuse.

## Sequence diagram View Single Risk.



## Description for fetching data for View Single Risk screen in ReactJS PoC

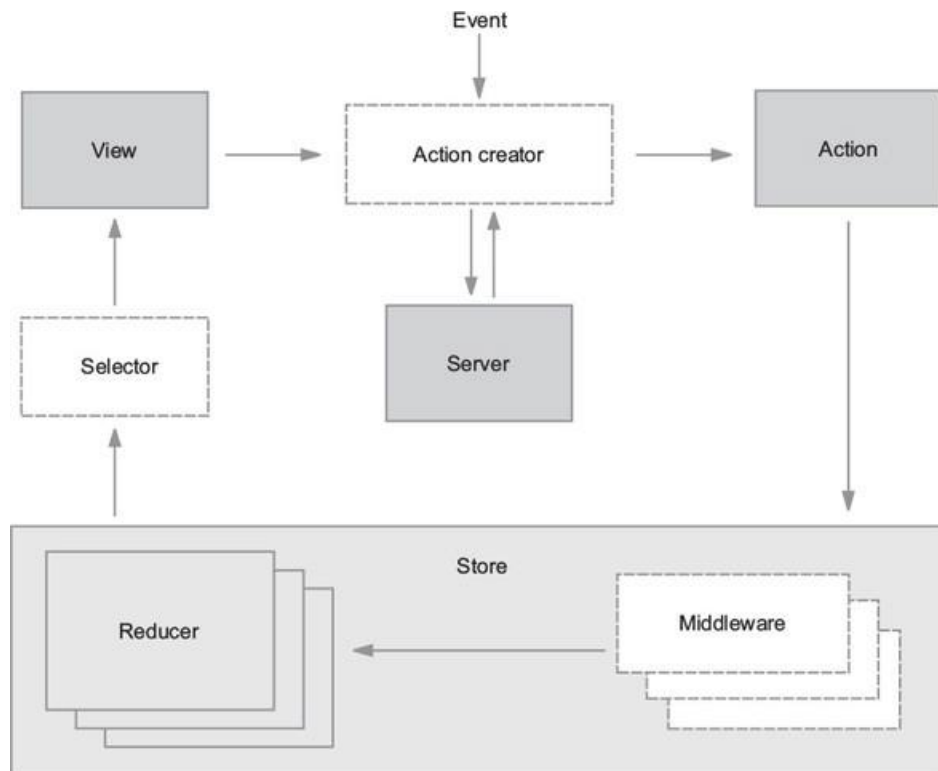
Here are the following participants in Authentication use case.

ViewSingleRiskCtrl, viewsingleriskActions, ReduxThunk, combinedReducer, Redux store, viewsingleriskService, RiskApi

1. **ViewSingleRiskCtrl** invokes bound method of **ViewSingleRiskCtrl** to dispatch to **getRisk** method to **viewsingleriskActions** passing riskid.
2. **viewsingleriskActions: getRisk** method dispatch **GET\_SINGLE\_RISK** action to ReduxThunk.
3. **ReduxThunk** in turn send **GET\_SINGLE\_RISK** action to combinedReducer.
4. **combinedReducer** dispatch **GET\_SINGLE\_RISK** to store which invokes matching action implemented by **viewsinglerisk** reducer it updates store and set **Loading** flag to true which is used by UI to show page loading image.
5. **viewsingleriskActions** further dispatches invokes Login method of **viewsingleriskService** which returns promise. Upon successfully resolving this promise **viewsingleriskActions: getRisk** method further dispatch **GET\_SINGLE\_RISK\_SUCCESS**, #2 to #4 repeats for **GET\_SINGLE\_RISK\_SUCCESS** which is used to set and process data of single Risk in shape (which is convenient to display on UI) in ReduxStore.
6. **viewsingleriskActions: getRisk** method further dispatch **alert(CLEAR)** action , #2 to #4 repeats for **alert(CLEAR)** which is used to clear error flag in ReduxStore. Error flag is used by **ToggleContainer** reusable component to clear out any **Alert** message shown to user in case any **RiskApi** call fails.
7. If **RiskApi** call fails due to invalid credentials or network error message flow take place as per error scenario mentioned in Sequence diagram.
8. If promise errors out **viewsingleriskActions: getRisk** method further dispatch **GET\_SINGLE\_RISK\_FAILURE**, #2 to #4 repeats for **GET\_SINGLE\_RISK\_FAILURE** which is used to set Error flag and message in ReduxStore. **ToggleContainer** uses error flag and error message to show Alert message to indicate Api method failure.

## Message flow within various classes in typical React & Redux based application

Message flow Diagram



## References

### ReactJS

<https://reactjs.org>

### Redux

<https://redux.js.org>

### Element React

<https://elemefe.github.io/element-react/index#/en-US/form>

### Create React App

<https://facebook.github.io/create-react-app/docs/getting-started>

### CoreUI for React

<https://github.com/coreui/coreui-free-react-admin-template>

### Enzyme

<https://airbnb.io/enzyme/>

### ReactJS and redux integration

<https://github.com/cornflourblue/react-redux-registration-login-example/blob/master/src/HomePage/HomePage.jsx>

### ReactJS children composition patterns

[https://medium.com/@martin\\_hotell/react-children-composition-patterns-with-typescript-56dfc8923c64](https://medium.com/@martin_hotell/react-children-composition-patterns-with-typescript-56dfc8923c64)

### ReactJS Redux testing using Jest and Enzyme

<https://alligator.io/react/testing-react-redux-with-jest-enzyme/>